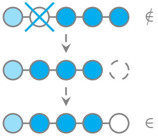
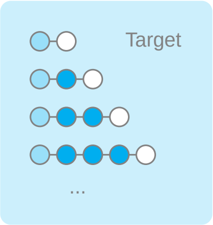
What do you do
if a computational object fails a specification?



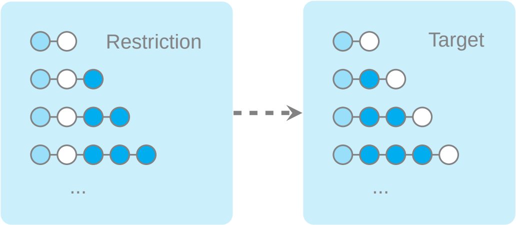


This problem has been studied over words:

1. “Regular repair of specifications”, in LICS 2011.
2. “The cost of traveling between languages”, in ICALP 2011.

We study this problem over XML Documents (trees).

What do you do
if a computational object fails a specification?



This problem has been studied over words:

1. "Regular repair of specifications", in LICS 2011.
2. "The cost of traveling between languages", in ICALP 2011.

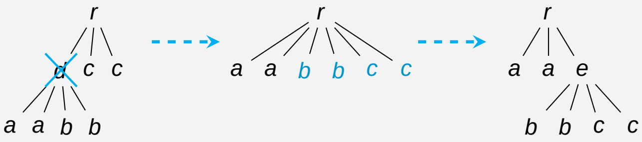
We study this problem over XML Documents (trees).

Can we repair each XML document with a uniformly bounded number of modifications?

Bounded Repair Problem

Example

R:	r → d c*	*T*:	r → a* e
	d → a* b*		e → b* c*
	a → EMPTY		a → EMPTY
	b → EMPTY		b → EMPTY
	c → EMPTY		c → EMPTY



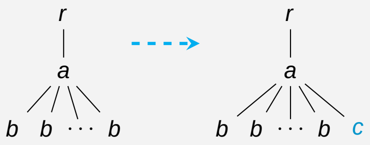
Can we repair each XML document with a uniformly bounded number of modifications?

Bounded Repair Problem

Example

R': r → a
a → b*
b → EMPTY

T': r → a
a → b*, c
b → EMPTY
c → EMPTY

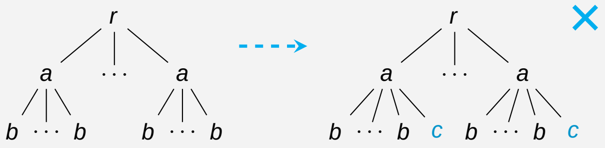


Can we repair each XML document with a uniformly bounded number of modifications?

Bounded Repair Problem

Example

R':	r → a*	T':	r → a*
	a → b*		a → b*, c
	b → EMPTY		b → EMPTY
			c → EMPTY



Can we repair each XML document with a uniformly bounded number of modifications?

Bounded Repair Problem

Example

R'':

r → a, d
a → a | EMPTY
d → b, c*
b → a
c → EMPTY

T'':

r → d, c*
d → a, a
a → a | b
b → EMPTY
c → EMPTY

? ? ? ?

In this talk,...
all about bounded repairability over trees

1. Effective characterization for every pair of regular tree languages.
   - Cyclic behavior of tree automata.
   - Covering mappings.
2. Algorithm to decide bounded repairability.
   - Complexity bounds.

Bounded repairability for regular tree languages

Cristian Riveros
Gabriele Puppis
Slawek Staworko

University of Oxford
February 2012

Outline

Setting

Characterization

From covering to repair

Complexity results

Concluding remarks

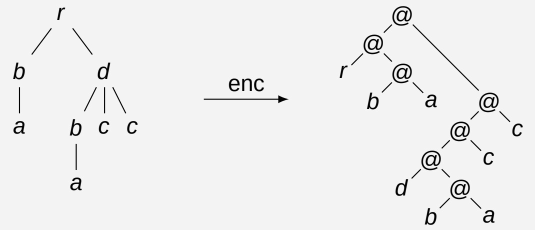
Outline

XML documents and (curried) trees

- XML documents = ordered unranked trees.
- Ordered unranked trees = curried trees.

$$
\begin{array}{ll}
f: & X \times Y \to Z \\
t: & a(t_1, t_2, t_3)
\end{array}
\quad \dashrightarrow \quad
\begin{array}{ll}
\mathrm{enc}(f): & f \to (X \to (Y \to Z)) \\
\mathrm{enc}(t): & (((a \mathbin{@} t'_1) \mathbin{@} t'_2) \mathbin{@} t'_3)
\end{array}
$$

Example

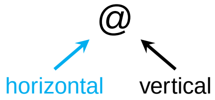


XML documents and (curried) trees

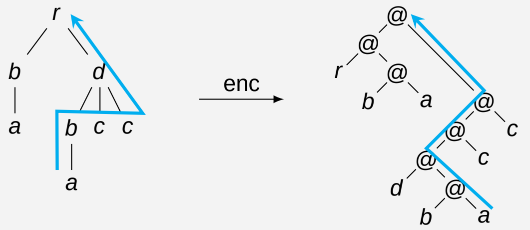
Definition

$$\mathrm{enc}(a) = a$$

$$\mathrm{enc}\left(\begin{array}{c} a \\ t_1 \cdots t_n \end{array}\right) = @\left(\mathrm{enc}\left(\begin{array}{c} a \\ t_1 \cdots t_{n-1} \end{array}\right), \mathrm{enc}(t_n)\right)$$



Example



Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

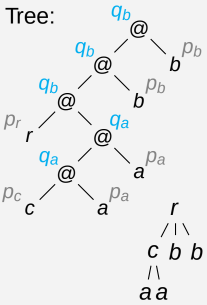
1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

Example

$$
\begin{aligned}
R:\ & r \to c\ b^* \\
& c \to a^+ \\
& a \to \mathrm{EMPTY} \\
& b \to \mathrm{EMPTY}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{R}:\ \delta(p_c, p_a) &\to q_a \\
\delta(q_a, p_a) &\to q_a \\
\delta(p_r, q_a) &\to \underline{q_b} \\
\delta(\underline{q_b}, p_b) &\to \underline{q_b} \\
\delta_0(a) &\to p_a \\
\delta_0(b) &\to p_b \\
\delta_0(c) &\to p_c \\
\delta_0(r) &\to p_r
\end{aligned}
$$

Tree:



Stepwise tree automata

Definition

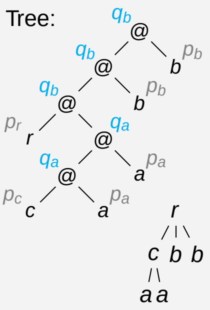
A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

Example

R:	r $\to$ c b*	$\mathcal{R}$:	$\delta(c, a) \to q_a$
	c $\to$ a$^+$		$\delta(q_a, a) \to q_a$
	a $\to$ EMPTY		$\delta(r, q_a) \to \underline{q_b}$
	b $\to$ EMPTY		$\delta(\underline{q_b}, b) \to \underline{q_b}$

Tree:



Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

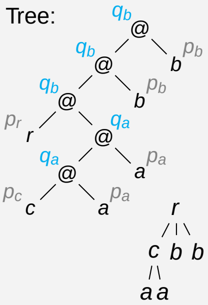
1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

Example

R: r → c b*
c → a$^+$
a → EMPTY
b → EMPTY

$\mathcal{R}$:
$$c \,@\, a \to q_a$$
$$q_a \,@\, a \to q_a$$
$$r \,@\, q_a \to \underline{q_b}$$
$$\underline{q_b} \,@\, b \to \underline{q_b}$$

Tree:



Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

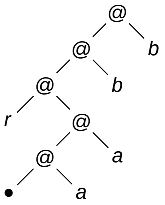
1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

We also define:

- tree language $L(\mathcal{A})$.
- contexts.
- concatenation between contexts:

$$C_1 \circ C_2.$$

- run of $\mathcal{A}$ on a context C from q.



Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

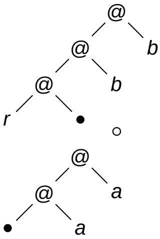
1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

We also define:

- tree language $L(\mathcal{A})$.
- contexts.
- concatenation between contexts:

$$C_1 \circ C_2.$$

- run of $\mathcal{A}$ on a context C from q.



Stepwise tree automata

Definition

A stepwise (tree) automata is a tuple $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$ such that:

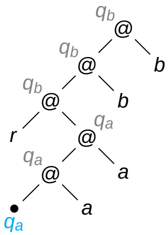
1. $\delta : Q \times Q \to 2^Q$ is the transition function,
2. $\delta_0 : \Sigma \to 2^Q$ is the initial function,
3. $F \subseteq Q$ is the final set of states.

We also define:

- tree language $L(\mathcal{A})$.
- contexts.
- concatenation between contexts:

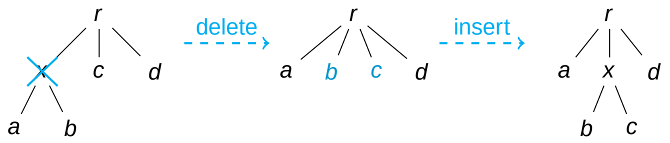
$$C_1 \circ C_2.$$

- run of $\mathcal{A}$ on a context C from q.



Edit operations over trees

Edit operations: deletion, insertion, and relabeling.



All operations have equal cost.

Definition

For trees t, t' and tree language T:

$$\mathrm{dist}(t, t') = \text{shortest sequence of operations that transform } t \text{ into } t'$$

$$\mathrm{dist}(t, T) = \min_{t' \in T} \{ \mathrm{dist}(t, t') \}$$

Bounded repair problem

Definition

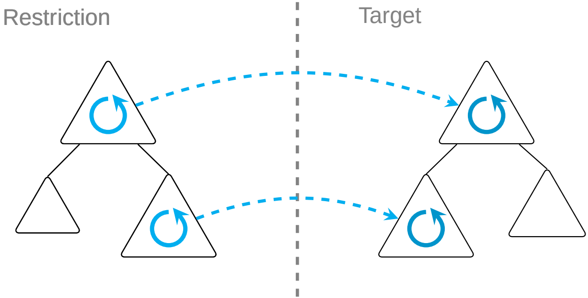
Given stepwise automata $\mathcal{R}$ (restriction) and $\mathcal{T}$ (target), determine if there exists a uniform bound $N \in \mathbb{N}$ such that:

$$\mathrm{dist}(t, L(\mathcal{T})) \leq N \quad \text{for all } t \in L(\mathcal{R})$$

Generalization of language containment.

Outline

How to repair trees? (intuition)

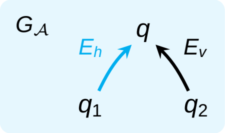


1. Cyclic behavior (Synopsis trees)
2. Mapping (Coverings)

Cyclic behavior of stepwise automata (components)

Definition

Given $\mathcal{A} = (Q, \Sigma, \delta, \delta_0, F)$, the transition graph of $\mathcal{A}$ is the graph $G_{\mathcal{A}} = (Q, E_h \cup E_v)$ such that for every $q \in \delta(q_1, q_2)$:

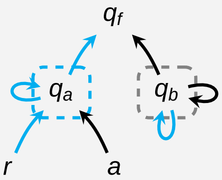


- $\mathrm{SCC}(\mathcal{A})$ is the set of strongly connected component X of $G_{\mathcal{A}}$.
- $L(\mathcal{A} \mid X) = \{C \in \mathrm{context}_\Sigma \mid \exists p, q \in X : q \in \delta(p, C)\}$

Example

$$\begin{array}{lcl} r & \to & a^* \cdot b \\ a & \to & \mathrm{EMPTY} \\ b & \to & b^* \end{array}$$

$$\begin{array}{rcccl} r & @ & a & \to & q_a \\ q_a & @ & a & \to & q_a \\ q_a & @ & b & \to & q_a \\ b & @ & b & \to & b \end{array}$$



 = horizontal,  = vertical

Synopsis trees

Definition

- A synopsis tree of $\mathcal{A}$ is a binary tree with labels in $\mathrm{SCC}(\mathcal{A})$.
- A primitive synopsis tree (PST) of $\mathcal{A}$ is a synopsis tree:
  1. every node respects the transition function of $\mathcal{A}$.
  2. every node has a different label from its children.

$$\begin{tikzcd}[column sep=small] & q \in X \arrow[dl, dash] \arrow[dr, dash] & \\ q_1 \in Y & & q_2 \in Z \end{tikzcd}$$

✓ $q \in \delta(q_1, q_2)$

✓ $X \neq Y$ and $X \neq Z$

- A basic synopsis tree (BST) of $\mathcal{A}$ is a synopsis tree:
  1. every node respects the transition function of $\mathcal{A}$.

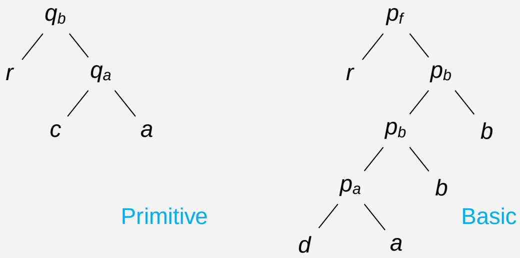
Example

R: $\mathrm{r} \rightarrow \mathrm{c}\ \mathrm{b}^*$
$\quad\ \ \mathrm{c} \rightarrow \mathrm{a}^*$

T: $\mathrm{r} \rightarrow \mathrm{d}$
$\quad\ \ \mathrm{d} \rightarrow \mathrm{a}^*\ \mathrm{b}^*$

$$\mathcal{R}:\quad \begin{array}{llll} c & @\ a & \rightarrow & q_a \\ q_a & @\ a & \rightarrow & q_a \\ r & @\ q_a & \rightarrow & \underline{q_b} \\ \underline{q_b} & @\ b & \rightarrow & \underline{q_b} \end{array}$$

$$\mathcal{T}:\quad \begin{array}{llll} d & @\ a & \rightarrow & p_a \\ p_a & @\ a & \rightarrow & p_a \\ p_a & @\ b & \rightarrow & p_b \\ p_b & @\ b & \rightarrow & p_b \\ r & @\ p_b & \rightarrow & \underline{p_f} \end{array}$$



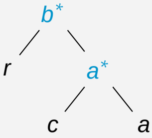
Example

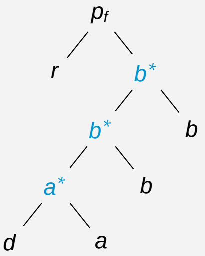
R: r → c b*
c → a*

T: r → d
d → a* b*

$$
\mathcal{R}:\quad
\begin{array}{llll}
c & @\ a & \rightarrow & q_a \\
q_a & @\ a & \rightarrow & q_a \\
r & @\ q_a & \rightarrow & \underline{q_b} \\
\underline{q_b} & @\ b & \rightarrow & \underline{q_b}
\end{array}
$$

$$
\mathcal{T}:\quad
\begin{array}{llll}
d & @\ a & \rightarrow & p_a \\
p_a & @\ a & \rightarrow & p_a \\
p_a & @\ b & \rightarrow & p_b \\
p_b & @\ b & \rightarrow & p_b \\
r & @\ p_b & \rightarrow & \underline{p_f}
\end{array}
$$





Synopsis trees approximate a regular tree language

Definition

The semantics $[\![\tau]\!]_{\mathcal{A}}$ of a synopsis tree τ of $\mathcal{A}$ is the language:

$$
\begin{aligned}
[\![X]\!]_{\mathcal{A}} &= \left\{ C \circ a \;\middle|\; C \in L(\mathcal{A}\,|\,X),\ a \in \Sigma \right\} \\
[\![X(\tau_1, \tau_2)]\!]_{\mathcal{A}} &= \left\{ C \circ (t_1 \mathbin{@} t_2) \;\middle|\; \begin{array}{l} C \in L(\mathcal{A}\,|\,X), \\ t_1 \in [\![\tau_1]\!]_{\mathcal{A}},\ t_2 \in [\![\tau_2]\!]_{\mathcal{A}} \end{array} \right\}
\end{aligned}
$$

with $X \in \mathrm{SCC}(\mathcal{A})$.

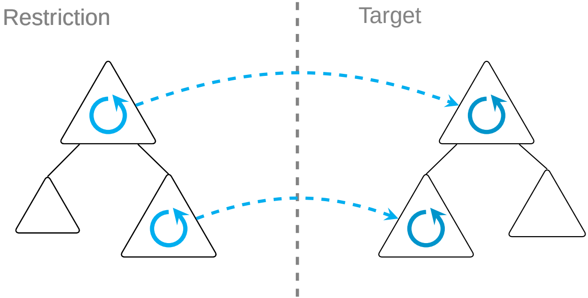
Lemma

For any stepwise automata $\mathcal{R}$ and $\mathcal{T}$:

$$
L(\mathcal{R}) \subseteq \bigcup_{\tau \in \mathrm{PST}(\mathcal{R})} [\![\tau]\!]_{\mathcal{R}} \quad \text{and} \quad \bigcup_{\tau \in \mathrm{BST}(\mathcal{R})} [\![\tau]\!]_{\mathcal{T}} \hookrightarrow_{\mathrm{BR}} L(\mathcal{T})
$$

We can represent the restriction and target with synopsis trees.

How to repair trees? (intuition)



1. Cyclic behavior (Synopsis trees)
2. Mapping (Coverings)

Coverings

Definition

Given two synopsis trees τ of $\mathcal{R}$ and σ of $\mathcal{T}$, we say that σ covers τ iff there exists a mapping λ from nodes of τ to nodes of σ:

1. λ preserves language containment of components,

$$L(\mathcal{R} \mid \tau(x)) \subseteq L(\mathcal{T} \mid \sigma(\lambda(x)))$$

2. λ preserves the post-order of nodes,

$$x \preccurlyeq_{\tau}^{\text{post}} y \text{ iff } \lambda(x) \preccurlyeq_{\sigma}^{\text{post}} \lambda(y)$$

3. λ preserves the ancestorship of vertical nodes,

$$x \preccurlyeq_{\tau}^{\text{anc}} y \text{ iff } \lambda(x) \preccurlyeq_{\sigma}^{\text{anc}} \lambda(y) \text{ with } x \text{ a vertical node}$$

for every non-trivial nodes x and y of τ.

Coverings

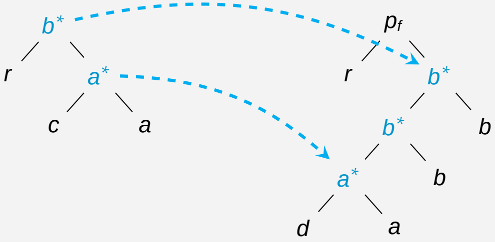
σ covers τ iff there exists a mapping λ from nodes of τ to nodes of σ:

1. λ preserves language containment of components,
2. λ preserves the post-order of nodes, and
3. λ preserves the ancestorship of vertical nodes.

Example

R: r → c b*
c → a*

T: r → d
d → a* b*



Main Characterization

Theorem

$L(\mathcal{R})$ is bounded repairable into $L(\mathcal{T})$ iff every primitive synopsis tree of $\mathcal{R}$ is covered by some basic synopsis tree of $\mathcal{T}$.

Two directions proof:

- From repair to covering.
- From covering to repair.

Outline

- Setting
- Characterization
- From covering to repair
- Complexity results
- Concluding remarks

From covering to repair

Lemma

For any stepwise automata $\mathcal{R}$ and $\mathcal{T}$:

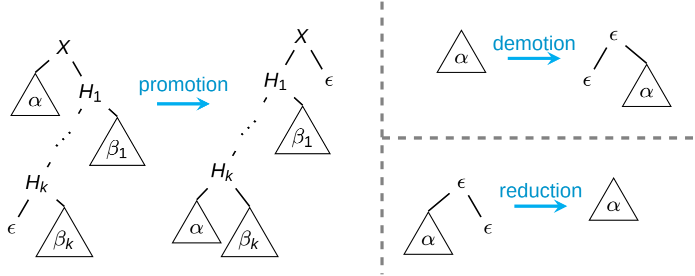
$$L(\mathcal{R}) \subseteq \bigcup_{\tau \in \mathrm{PST}(\mathcal{R})} [\![\tau]\!]_{\mathcal{R}} \quad \text{and} \quad \bigcup_{\tau \in \mathrm{BST}(\mathcal{R})} [\![\tau]\!]_{\mathcal{T}} \hookrightarrow_{\mathrm{BR}} L(\mathcal{T})$$

It only left to show that: $[\![\tau]\!]_{\mathcal{R}} \hookrightarrow_{\mathrm{BR}} [\![\sigma]\!]_{\mathcal{T}}$.

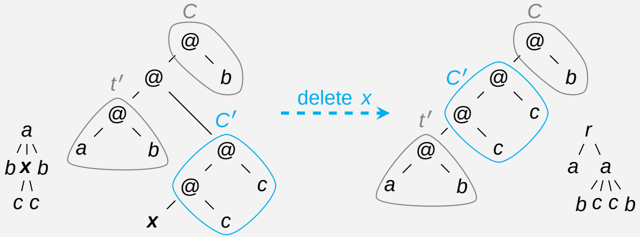
Outline of the proof:

1. Normal form over synopsis tree.
2. Covering implies isomorphic normal forms.
3. Set of operations to transform any synopsis tree into its normal form.
4. Operations over synopsis tree preserves bounded repairability.

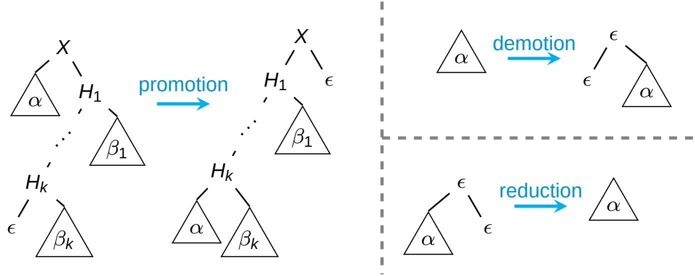
Synopsis tree operations



Remark.



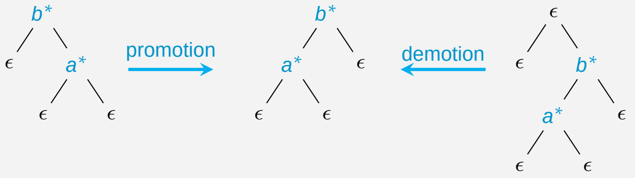
Synopsis tree operations



Example

R: r → c b*
c → a*

T: r → d
d → a* b*



Outline

Complexity results

Given stepwise automata $\mathcal{R}$ and $\mathcal{T}$:

- $\#$-primitive synopsis trees is $\leq 2^{|\,\mathrm{SCC}(\mathcal{R})|}$.

Lemma

Given a primitive synopsis tree τ of $\mathcal{R}$, the basic synopsis tree σ that covers τ is at most of size $2 \cdot |\tau| \cdot |\,\mathrm{SCC}(\mathcal{T})|$.

Complexity results

Given stepwise automata $\mathcal{R}$ and $\mathcal{T}$:

- $\#$-primitive synopsis trees is $\leq 2^{|\mathrm{SCC}(\mathcal{R})|}$.
- $\#$-basic synopsis trees is $\leq 2^{|\mathrm{SCC}(\mathcal{R})|+1} \cdot |\mathrm{SCC}(\mathcal{T})|$.

Algorithm:

1. Universally-guess a PST τ of $\mathcal{R}$ of size $2^{|\mathrm{SCC}(\mathcal{R})|}$.
2. Existentially-guess a BST σ of $\mathcal{T}$ of size $2^{|\mathrm{SCC}(\mathcal{R})|+1} \cdot |\mathrm{SCC}(\mathcal{T})|$.
3. Existentially-guess a covering function λ.
4. Checks that λ is a covering of τ by σ.

Proposition

The bounded repairability problem for regular tree languages is:

- in Π_2^{EXP}.
- EXPTIME-hard.

Complexity results for DTDs

We consider restrictions several restrictions:

- deterministic DTDs
- non-recursive DTDs

Proposition

The bounded repair problem between languages represented by deterministic DTDs is PSPACE-hard, even for non-recursive DTDs.

Better complexity results by fixing the alphabet.

Proposition

The bounded repair problem:

- for DTDs over a fixed alphabet is in EXPTIME.
- for deterministic DTD over a fixed alphabet is in Π_2^P.

Outline

Concluding remarks

- Effective characterization for every pair of regular tree languages.
- Algorithm to decide bounded repairability.
- Many open problems about the complexity.
- Future work: bounded streaming repair.

Bounded repairability for regular tree languages

Cristian Riveros
Gabriele Puppis
Slawek Staworko

University of Oxford
February 2012